A Simulation Environment for Job Scheduling on Distributed Systems

J. Santoso[1,2], G.D. van Albada[2], T. Basaruddin[3], and P.M.A. Sloot[2]

[1] Dept. of Informatics, Bandung Institute of Technology
Jl. Ganesha 10, Bandung 40132 Indonesia
[2] Dept. of Computer Science, University of Amsterdam Kruislaan 403,
1098 SJ Amsterdam The Netherlands
[3] Dept. of Computer Science, University of Indonesia,
Kampus UI Depok, Jakarta Indonesia

Abstract. In this paper we present a simulation environment for the study of hierarchical job scheduling on distributed systems. The environment provides a multi-level mechanism to simulate various types of jobs. An execution model of jobs is implemented to simulate the behaviour of jobs to obtain an accurate performance prediction. For parallel jobs, two execution models have been implemented: one in which the tasks of the job frequently synchronise and effectively run in lock step and a second in which the tasks only synchronise at beginning and end.
The simulator is based on an object approach and on process oriented simulation. Our model supports an unlimited number of workstations, grouped into clusters with their own local resource manager (RM). Work is distributed over these clusters by a global RM. To validate the model, we use two approaches, analysing the main queueing systems and experimenting with real jobs to obtain the actual performance as a reference.

1 Introduction

Workstations are used as computing resources for various applications because of their computing power and cost-effectiveness. Integrating several workstations using a high speed network into a cluster can create a high performance computing environment, and allows users to share expensive resources. Combining several clusters into a wide-area system further increases the potential for of resource sharing. However, an efficient resource management system is needed to utilise the available resources optimally. The main resource management activities are allocating the resources for the applications and scheduling the applications to satisfy predefined performance criteria.

Designing an appropriate scheduling approach for such an environment is an open research question. Scheduling is an NP-hard problem even for homogeneous resources. A good scheduler should be able to satisfy the given constraints, to achieve the scheduling objectives, and to operate with low-overhead. The objectives of the scheduling can be diverse, to obtain a better load sharing, load balancing and to meet user performance expectations. In this case, no optimal schedule can be achieved easily.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2329, pp. 653–662, 2002.

The accepted way to study scheduling is through a combination simulation and experimentation. Hence, we have built a simulation environment that allows us to study the scheduling performances of parallel programs in a multi-cluster environment. Simulation models have been widely used to study computer systems and network properties [5, 8]. Damson [6] uses a simulation to predict the performance and scalability properties of a large scale distributed object system. Another example is HASE [1] that is used to simulate computer architectures at multiple levels of abstraction, including hardware and software. The model supports various operation modes, e.g. design, model validation, build simulation, simulate system, and experiments.

Our work differs from the above studies in the following aspect. We do not use our simulation model as the subject of study, but use it to support our study on job scheduling. Our model does not represent a specific system. The level of detail in the simulation is an important parameter that influences the accuracy of the results. Therefore, we decompose the system into several subsystems, and each subsystem can be modelled separately to the required level of detail. We have validated the model by comparing simulation results with the actual performance of PVM jobs and and by comparing with queueing theory.

The goal of our simulation system is to simulate high performance computing (HPC) and high throughput computing (HTC) environments. For HPC we evaluate the efficiency of the system, e.g. on basis of the response time and the slowdown of jobs that are collected when the jobs leave the system. For HTC, the number of finished jobs for a simulation run and total amount of work can be used as a criterion.

This paper is organised as follows. In section 2, our model of distributed system will be described as well as several scheduling strategies used. The functional design of our simulation environment is discussed in section 3, and the implementation is explained in section 4. In section 5, we describe the model validation and in section 6, we come to our conclusions.

2 Conceptual model

2.1 Distributed system

We restrict our model to distributed systems consisting of clusters of workstations(COWs). A distributed system is an aggregation of (possibly heterogeneous) resources, in clusters at geographically distributed sites, e.g an I-WAY system[4]. A cluster is assumed to be an autonomous system with its own policies in managing resources and in scheduling jobs.

The structure of the resource organisation determines the model of its scheduling system. Federated scheduling is used by Legion [10] to exploit remote resources in a wide-area system with the objective to provide high performance and capabilities not available locally. The Globus meta-computing environment [3] implements a scheduling framework that allows for co-allocation, i.e. the co-eval distribution of a job over multiple clusters. In Globus and Legion local cluster

managers collaborate to realise global scheduling implicitly. Another approach is an hierarchical organisation [9] that divides the resource management system into global and local levels. The global management system explicitly provides several functionalities that are not available in the local managements system, e.g. global resource monitoring and wide-area scheduling. Our scheduling model is based on this hierarchical architecture.

In our simulation model, we define a COW as a collection of computing resource entities (workstations). Each entity has a set of attributes to indicate the characteristics of the workstation, like processing speed. Each entity has its own CPU scheduler, responsible for allocating processor time to tasks. Each COW has a local scheduler that assigns tasks to workstations in that cluster.

To model the applications running on the distributed system, we consider various synthetic jobs consisting of a number of tasks. We assume a sequential job to have only one task, whereas parallel jobs have at least two tasks. Each task comprises computation and communication phases. All tasks live for the entire duration of the job and represent an equal amount of work.

The synthetic parallel jobs can model a variety of communication and synchronisation patterns. Standard models include the master slave and lock-step SPMD paradigms. In the lock-step SPMD paradigm, the tasks of a parallel job synchronise frequently before resuming the next computation phase. In the master-slave model, the master task sends the work to all slave tasks and collects the output when the computation finishes.

After a job has been created, it is assigned to a cluster by the global scheduler. We will not consider the actual implementation of such a scheduler, but we will assume the existence of an abstract global scheduler. Multiple scheduling criteria can be considered, e.g. clusters are selected for specific applications based on their performance, reliability, connectivity and available resources.

2.2 Hierarchical scheduling

Hierarchical resource management divides the resource management system into several layers. We distinguish a global scheduling layer, a local scheduling layer, and a CPU scheduling layer. The global and local schedulers have different functionalities in several respects. The global scheduler is responsible for assigning arriving jobs to individual clusters by considering the imposed global policies to select the best candidate clusters. The local scheduler in a cluster is responsible for distributing tasks of jobs to a set of resources in its cluster. On those resources, the tasks are scheduled by the CPU scheduler that simulates the operating system scheduler. The local scheduler applies its local policies that need not depend on the external system. For an optimal performance, both global and local schedulers should cooperate and exchange information.

The goal of our scheduler is to optimise one performance measure, while keeping other measures within reasonable bounds. Initially, we will use crude schedulers, like SJF and determine their performance as measured against a large number of criteria. Later, we will experiment with more sophisticated schedulers that consider multiple criteria simultaneously.

Global scheduling The global scheduler is designed for top-level scheduling purposes. Newly arriving jobs will be scheduled by the global scheduler to available clusters. If no cluster is available, the jobs are placed in a global queue. When the resources are available the jobs are dispatched by the global scheduler according to some optimality criteria based on the available information on cluster capacity, load and/or job characteristics.

Local scheduling The local scheduler maintains a single queue to store jobs if they cannot be scheduled immediately. When resources become available, the waiting jobs will be dispatched. Each job consists of a number of tasks that does not vary throughout the lifetime of a job. The local scheduler allocates each task to a resource. By default, the tasks of a parallel job are distributed evenly among the available resources, starting from the least loaded resource. Round-robin scheduling as used by PVM is also modelled. .

The CPU scheduler The runnable tasks on each node are dispatched in a round-robin fashion. As the tasks in a parallel program are assumed to synchronise after every time step, a task becomes runnable again only when all other tasks in the job have advanced to the same point. Gang scheduling can also be simulated. The communications delay is assumed to be a constant value.

3 Functional Design of the Simulation Model

In principle, the simulation model and the modelled system have a similar structure. The components of the simulation model are derived from the components of the system to obtain a representative model and to improve the validity of the model. We have chosen to use an object oriented approach. The correspondence between objects and components allows the object oriented model to mimic the real system. Object oriented models can easily be expanded; a large system can be built incrementally, new object classes can be created from existing classes.

3.1 Interactions and object interfaces

The simulation is based on discrete events and is process oriented. In this approach the management of events is implicit in the management of the processes. We distinguish passive and active objects. The latter are responsible for the control of the internal simulation activities such as event scheduling, time advance and collecting statistics. These objects are associated with an independent thread of control at creation time and can generate events. In our model, there are two classes of active objects, the arrival class and the resource class. When a simulation is started all persistent objects are initialised. Next, the arrival process is activated, Fig. 1. The first task arriving at a resource will activate it. The various objects interact through method invocation and simulation primitives.

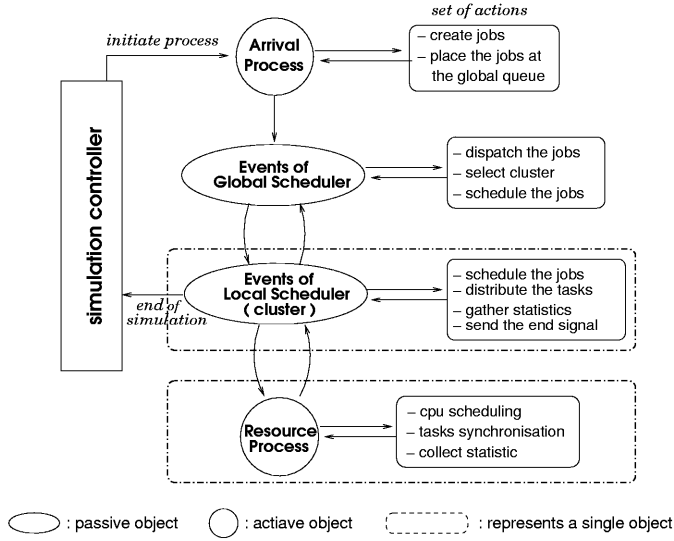


Fig. 1. Simulation process

3.2 Objects and Functionalities

To build our simulation environment, we have defined eight basic classes that represent the actual functionalities of the system.

Class Job and Class Task A *job* object is a passive object. It has a set of attributes describing its characteristics and states. Some attributes are initialised by the *arrival* object when the *job* instance is created; the other attributes are set during its life time in the simulation run, e.g. job response time and start of execution. The information is retrieved when the job leaves the system.

The tasks of a parallel job are also modelled as objects and they are created and initialised by the cluster object when the job starts execution. During the execution the tasks are controlled and scheduled by the resource object.

Class Arrival The *arrival* object generates synthetic jobs of various types until the end of the simulation. Jobs with their attributes are generated individually or in batches according to a stochastic distribution. The generated jobs are placed in the global queue waiting to be scheduled.

Class Resources A resource object represents a computing resource. It has a set of attributes that identifies its characteristics, e.g. speed, maximum load capacity, time quantum length, and synchronisation delay. The values are initialised by the cluster object. A resource object is activated by a cluster object

when tasks are scheduled to this resource and it is in the idle state. While active, the resource object manages the task queue that represents the load of the resource. The tasks in the queue are scheduled in round robin fashion by the *CPU scheduler*. The resource object also coordinates the synchronisation of tasks in a parallel job. All tasks are stored in the round-robin queue. When the task at the head of the queue is blocked, the CPU is given to the next task. The blocked task remains at the head of the queue, so that it can be run immediately when it is unblocked. This mimics the increased priority of I/O bound processes in commonly used CPU schedulers.

Class Global scheduler The global scheduler object schedules jobs generated by the arrival process. The scheduler matches the requirement of the job in the global queue to the resources available in the local cluster. The global scheduler can manage one or more job queues. The details of matching and the resulting schedule depend on the global policy. If suitable jobs are found, they can be further scheduled in that cluster.

The global scheduler object is a passive object; the events occurring in this object are triggered when a job arrives and when the resource states change. Scheduling for a new arrival event is performed as follows. When a new job arrives, it is added to the appropriate queue. Next, all clusters are queried about the availability of resources and the job is scheduled, if possible.

Class Cluster A *cluster* is responsible for scheduling jobs, distributing tasks to the resources, and monitoring the state of the resources. All events occurring in this object are triggered by external method invocations from the global scheduler object and the resource objects. Scheduling is initiated when jobs are received from the global scheduler and when resources become available. The cluster maintains one local queue to store jobs that cannot be scheduled immediately. The jobs in this queue are dispatched when resources become available.

Class Job queue The global scheduler and cluster object use *job queue* objects to manage their job queues. Job queue objects have dispatching primitives that allow the global and cluster schedulers to apply various scheduling policies.

Class Task queue A *task queue* object manages the running tasks in the resources. This object has a set of methods to support CPU scheduling, e.g. dispatch the tasks, suspend running tasks, and modify the task state (*ready, blocked, waiting*).

4 Implementation

The simulation environment was developed using the C++SIM simulation package [2]. C++SIM provides an extensive set of simulation and statistical primitives. In a simulator it is important to provide for independent streams of random

numbers for each stochastic variable. We added this feature in our system, based on the mtGen generator from Matsumoto [7]. A different random stream can be generated easily by varying the initial seed. The various independent streams are used to generate the random variates of the expected execution time, the job's inter-arrival time, and the job type distribution.

Simulation control and data collection The simulation environment provides a mechanism to collect the outputs within a sub-time-interval in a single long run. This enables us to make replications of runs without re-initialising the simulation parameters. Only the values within a sub-interval are sampled. Sampling periods can be preceded and separated by settling periods in which no statistics are obtained.

Configuring the model The model has a set of configurable parameters. The resource configuration is read from a file during the initialisation. The file describes the number of resources, the characteristics of each resource, and the organisation of those resources. Other parameters are entered as program arguments, e.g. random seed, mean inter-arrival time, scheduling policies and number of simulated jobs. Default values are provided when necessary.

5 Validation

The aim of validation is to ensure that the results of a simulation run can be accepted with enough confidence. It is not possible to prove that the model is absolutely correct. We have used two independent approaches for model validation. In the first approach, we analyse the queueing system at the local and global level for various configurations to verify its consistency with theory and to ensure that the schedulers operate correctly.

In another approach, we validate the model by comparing the simulation results with actual runs of PVM jobs. The objective here is to ascertain that our scheduling models provide an adequate representation for the more complex scheduling on our Unix system. For this purpose, we create the real PVM jobs with the same characteristics as those of the synthetic jobs used in simulation.

5.1 Queueing System : $M/M/4$ and $M/G/1$

For an $M/M/4$ queueing system, we compare the statistics of the job response times computed using the standard theory with those obtained from simulation for various arrival rates with a constant mean execution time. The results are shown in Fig. 2. We have used the following parameter values. The mean execution time $(1/\mu)$ is four unit times; the number of servers is four, the number of jobs for one simulation run is 4000. A single simulation result is obtained by performing 16 simulation runs using different random streams. This allows us to also obtain error estimates and standard deviations of the response times.

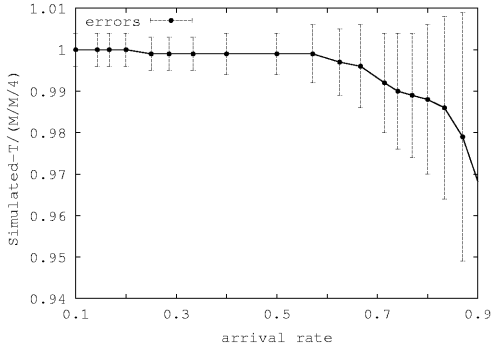


Fig. 2. The response time vs. arrival rate for $M/M/4$. The measured values have been normalised by dividing them by the predicted values from theory. The error bars are derived from the variations between the 16 runs contributing to a single measurement. As the same random streams were used for each arrival rate, the measurements at different arrival rates are not independent.

The results for the $M/M/4$ queueing system show that the simulation results are consistent with theory. From this we may conclude that the code has not made gross scheduling and accounting errors and that the random generators indeed approximate the desired distributions (this had, of course been verified before using more stringent tests).

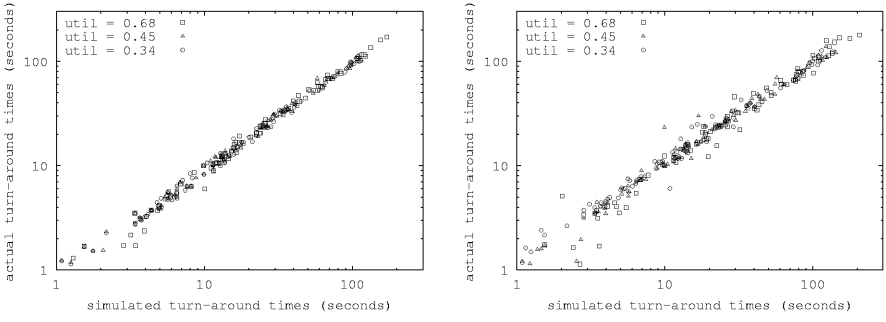
We also performed a partial test of the treatment of multi-task jobs. As queueing theory considers only sequential jobs, we treat N-task jobs as sequential jobs using N times as much CPU time. Jobs are dispatched using an FCFS policy to a single workstation, and the workstation is allowed to hold one job at a time without preemptive scheduling. As we use a mix of sequential and parallel jobs, this results in an $M/G/1$ scheduling problem, for which the turn-around time can be easily calculated from theory. The simulation results for this configuration (not shown) were also found to be consistent with theory.

5.2 Validation with the PVM jobs

As the goal of our simulation is to describe the behaviour of real jobs in real systems, we have compared the turn-around times predicted by our simulator with those of a batch of simple, real parallel jobs run on a COW. We used PVM jobs (*master-slave and lock-step SPMD models*) with one to four tasks. The main difference between two models is the communication frequency between the tasks: the lock-step SPMD model synchronises at every time step (1 second); master-slave programs are assumed to communicate only at the beginning and the end. In both cases the tasks monitor their own CPU time use in order to ensure consistency with the simulation. Equal numbers of one, two, three, and four-task jobs were generated. The jobs were submitted directly to the workstation with

the inter arrival time following an exponential distribution with means of 10, 15 or 20 seconds. We let the PVM round-robin task scheduler distribute the tasks to the workstations. The mean execution time of jobs was 10 seconds. The execution environment consists of four SUN Ultra-5 workstations.

The experimental results are shown in Fig. 3-a and Fig. 3-b. The communication delay of interacting tasks in the actual run has a high variance due to external factors, e.g background load and traffic intensity in the network. When the utilisation of resources is high, these effects are much more obvious, especially for the small jobs (see Fig. 3-b). Master-slave, which has less communications is modelled much more accurately than SPMD.



(a) Master-slave model

(b) Lock-step SPMD model

Fig. 3. Performance of simulation and actual run

6 Conclusion

We have built a simulation environment to study job scheduling strategies on a distributed system consisting cluster of workstations. The system is designed and built using an object oriented approach. Our goal is to obtain a better understanding of the interaction of resource elements and scheduling flows. The system incorporates some specific features to provide more flexibility.

- It supports several types of synthetic jobs, including serial and parallel.
- The global scheduler supports a multi queueing system.
- The resource structure and organisation can be configured easily.
- A variety of scheduling policies can be applied.
- A low level job simulation (*CPU scheduler*) is implemented.

With this model, we can study job and task scheduling simultaneously at a low cost, rather than experimenting in the actual system. The behaviour of the

running jobs can be observed , which is useful in analysing the performance of the schedule. The individual resource performance can be monitored for dynamic scheduling and load balancing purpose. On the other side, there are simplifications that cause some outputs are not comparable with the actual performance.

- The synthetic jobs are simplified.
- The resources are limited to cluster of workstations.
- The effects of other resources (e.g. I/O, memory cache, network bandwidth, files, and file servers) are not addressed.

Acknowledgements

We gratefully acknowledge support for this work by the Dutch Royal Academy of Science (KNAW) under grant 95-BTN-15. We wish to thank Yohanes Stefanus of the University of Indonesia for fruitful discussions.

References

[1] P. Coe and et. al. Technical note: A hierarchical computer architecture desi gn and simulation environment. *ACM Trans. on Modeling and Computer Simulation*, 1998.

[2] *C++SIM Simulation Package*, Univ. of Newcastle Upon Tyne, 1997. `http://cxxsim.ncl.ac.uk/`.

[3] K. Czajkowski, I. Foster, N. Karonis, C. K. lman, and et. al. A resource management architecture for metacompting systems. In *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, pages 62–82, Berlin, 1998.

[4] T. A. DeFanti, I. Foster, M. E. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide area visual supercomputing. *International Journal of Supercomputer Applications and High Performance Computing*, 10(2/3):123–130, 1996.

[5] M. Falkner, M. Devetsikiotis, and I. Lambadaris. Fast simulation of networks of queues with effective and decoupling bandwidths. *ACM Transaction on Modeling and Computer Simulation*, 9(1):45–58, January 1999.

[6] S. Frolund and P. Garg. Design-time sim. of a large-scale distr. object system. *ACM Trans. on Modeling and Computer Simulation*, 8(4):373–400, October 1998.

[7] M. Matsumoto and T. Nishimura. "mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator". In *ACM Transactions on Modeling and Computer Simulation*, volume 8. ACM, January, 1998.

[8] P. Spinnato, G. van Albada, and P. Sloot. Performance prediction of n-body simulations on a hybrid architecture. *Computer Physics Communications*, 139:33–34, 2001.

[9] A. van Halderen, B. Overeinder, P. Sloot, R. van Dantzig, D. Eperma, and M. Livny. Hierarchical resource management in the polder metacomputing initiative. *Parallel Computing*, 24(12/13):1807–1825, November 1998.

[10] J. B. Weissman and A. S. Grimshaw. A federated model for scheduling in wide-area systems. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pages 542–550, Syracuse, NY, Aug. 1996.